Implementation of BEDSOCS: an interactive simulation language

A. F. Eidelson* and I. J. Robinson†

BEDSOCS, an interactive digital computer language that is designed to allow the easy solution of problems described by ordinary differential equations, is used to run PHYSBE (a benchmark program that simulates the human circulatory system). A comparison chart compares BEDSOCS with other simulation systems in time required to execute PHYSBE.

(Received January 1979)

Bradford EDucational Simulation language fOr Continuous Systems (BEDSOCS), is an interactive digital computer language designed to allow the easy solution of problems described in terms of ordinary differential equations. The language uses Dartmouth BASIC as the procedural language and, like most BASIC systems, operates interpretively. BEDSOCS has such advanced features as automatic equation sorting, variable-step integration and graphic displays. The equations describing the simulation need not be represented in terms of block diagrams, but rather are entered much like *ordinary differential equations*. Above all, because BEDSOCS uses an interpreter, there is no compilation time. This allows users to write and debug programs quickly at the cost of longer execution time.

BEDSOCS, written by Geoffrey Brown (1973) at the University of Bradford, can be implemented on an HP2100 computer having 16k words of memory. The system uses HP BASIC as a subset and supports floating point firmware and a Tektronix storage display. Many options are available for implementing BEDSOCS. However, the authors used a single user version distributed on paper tape. Using BASIC as a subset, BEDSOCS partitions a program into a *control region*, which contains BASIC statements, and a *dynamic region*, which describes the representation of the system to be simulated. In the control region the user can program in BASIC to establish initial conditions, set control variables and perform general computation and input/output. The dynamic region contains the representation of the set of differential equations (see **Fig. 1**).

The control region is divided into the *initial region* and the *terminal region*. The initial region precedes the dynamic portion of the BEDSOCS program and is used to set initial conditions and control variables to be used in the dynamic region. The terminal region is executed when the simulation is completed. This region is generally used to perform final calculations, print results, or loop back for iterative differential equation solving runs. The dynamic region of a BEDSOCS program contains the 'representation set'. This is a set of equations and procedure blocks representing the simulation to be performed. **Fig. 2** shows a typical procedure block, enclosed by the 'PROCED' and 'PROEND' statements. The 'PROCED' statement has the form:

⟨statement no.⟩ PROCED ⟨output list⟩ = ⟨input list⟩

where the ⟨input list⟩ contains the dependent variables used within the procedure block and the ⟨output list⟩ contains the dependent variables which are assigned values by the procedure block. Within a procedure block, the user may describe non-linear functions or other operations involving any BASIC statement just as in the control region. There may be any number of procedure blocks within a BEDSOCS program. However, a variable may not occur in the output list of two procedure blocks. Inputs and outputs of procedure blocks are sorted just like those of a defined variable equation.

Equations of the representation set differ from BASIC statements in their location and form. These statements are not preceded by the 'LET' identifier of BASIC and they are automatically sorted into a BASIC program which evaluates derivatives and then updates state variables (integration). For example, the following differential equation may be expressed in BEDSOCS with the aid of the 'DER' identifier.

$$dx^2/dt^2 - 5x + 3 = y$$

In BEDSOCS

300	DER $Z = Y - 5 * X + 3$
310	DER $X = Z$
or	
300	$W = 5 * X - 3$
310	DER $Z = Y - W$
320	DER $X = Z$

These simple equation statements combined, if necessary with procedure blocks, make it very easy to describe complex systems. Since BEDSOCS is interpretive, it is executed without compilation. When the user types 'RUN', BEDSOCS checks the program structure, creates the symbol table, sorts the representation set and initialises the independent variable or variables to zero. As this initial setup is performed, errors are displayed on the user's console with the number of the line where the error occurred. Execution continues procedurally

```
 10 LET K = 1                                      control region (initial)
 20 LET X = 0
 30 LET V = -0.2
 40 LET D =  0.5
 50 LET M = 1
 70 PRINT "DIST","VEL.","ACCEL.","FORCE"
220 DYNAMIC                                        dynamic region
400  EQUATIONS                                     equations section
600   PROCED  F = X,K,D                            representation set
610    IF X    D THEN 650
620    IF X   -D THEN 670
630    LET F = 0
640    GOTO 680
650    LET F = K*(X-D)
660    GOTO 680
670    LET F - K*(X+D)
680   PROEND
690    A = -F/M
700   DER V = A
710   DER X = V
800  EQUEND
900  PRINT X,V,A,F
950 DYNEND
975 PRINT "TERMINATION SECTION"                    control region (terminal)
985 END
```

Fig. 1 The regions of a typical BEDSOCS program (Brown, 1973)

```
600    PROCED  F   X,K,D
610     IF X >  D THEN 650
620     IF X <-D THEN 670
630     LET F = 0
640     GOTO 680
650     LET F = K*(X-D)
660     GOTO 680
670     LET F = K*(X+D)
680    PROEND.
```

Fig. 2 A typical BEDSOCS procedure block

*159 Frederick Street, Peekskill, New York 10566, USA, employed by IBM System Communication Division, Kingston, New York, USA.
†Employed by Motorola Microsystems, Mesa, Arizona, USA.

Downloaded from https://academic.oup.com/comjnl/article/23/3/233/375185 by guest on 10 April 2024

through the control region until the dynamic region is encountered. At that time, the integration routines are initialised and execution continues into the dynamic region. The dynamic

```
  1 REM: PROJ. OVER WALL
 10 READ G,F,G1,G2,V0,A0,E0
 30 DATA 9.8,1.00000E-02,.6,.6,50,45,.5
 40 PRINT "RANGE";
 45 INPUT R
 48 LET @[2]=@[4]=@0 0
 49 LET @[1]=@[3]=R
 55 PRINT "DIST. & HEIGHT OF WALL";
 60 INPUT W,H
 77 LET A0=ATN(1)A0/45
100 LET V1=V0*OS(A0)
105 LET V2=V0*SIN(A0)
110 LET V=SQR(V1↑2+V2↑2)
125 LET A=ATN(V2/V1)
130 LET V0=V
135 LET A0=A
140 LET X=Y=T=0
150 LINE W,0,W,H
200 DYNAMIC
210  EQUATIONS
220   INDVAR T
223   DER V=-G*SIN(A)-F*V 2
227   DER A=-(G/V)*COS(A)
230   DER X=V*COS(A)
240   DER Y=V*SIN(A)
250   EXIT (X >= W) ───────────┐
260   EXIT (Y <= 0 AND X>0) ─┐ │
450   DISPLAY X;Y            │ │
550  EQUEND                  ↓ ↓
560  IF Y <= 0 GOTO 600 ◄────┴─┘
570  LET Z=Y
580 DYEND
600 LET E1=R-X
610 LET E2=H-Z
620 IF ABS(E1)+ABS(E2)<E0 THEN 650
630 LET V1=V1+G1*E1
635 LET V2=V2+G2*E2
640 GOTO 110
650 PRINT "TIME","HEIGHT AT WALL","RANGE","SPEED","ANGLE"
660 PRINT T,Z,X,V0,A0*45/ATN(1)
```

Fig. 3 A BEDSOCS program to calculate the path of a projectile

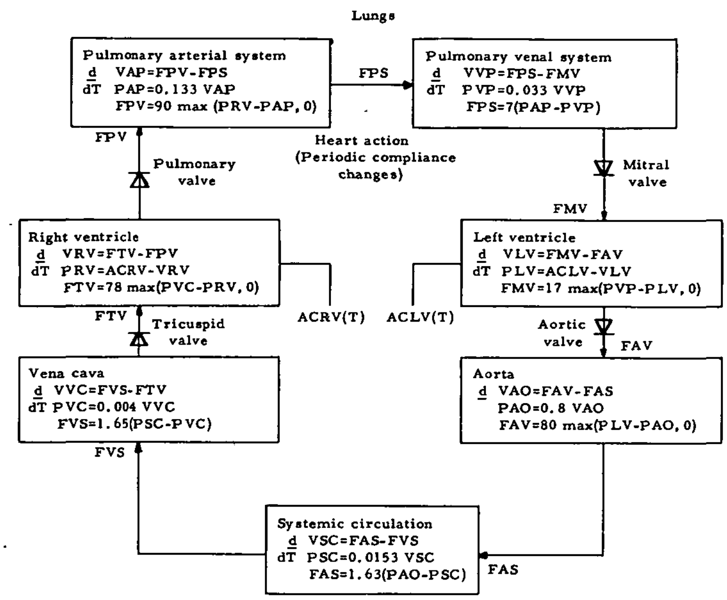

Fig. 4 A block diagram of the PHYSBE model (Korn and Wait, 1976)

Table 1

```
      PAGE  0001          *PHYS3          19 MAY 1978   00:10:12

  20 REM ************* PHYSBE *****************
  40 REMARK
  60 REM
  80 REM  VARIABLE EQUIVALENCES
 100 REM
 120 REM   A  ACLV     B  ACRV
 140 REM   V1 VAO      P1 PAO      F1 FAS
 160 REM   V2 VAP      P2 PAP      F2 FAV
 180 REM   V3 VLV      P3 PLV      F3 FMV
 200 REM   V4 VPV      P4 PRV      F4 FPS
 220 REM   V5 VSC      P5 PSC      F5 FPV
 240 REM   V6 VVC      P6 PVC      F6 FTV
 260 REM   V7 VVP      P7 PVP      F7 FVS
 265 REM
 266 REM
 267 REM
 270 REM  DIMENSION A AND B FOR THE LOOKUP TABLE TO REPRESENT
 271 REM   THE HEART PUMPING.  IT IS NECESSARY TO WRAP AROUND
 272 REM   TO 30 RATHER THAN STOPPING AT 25 TO BE CERTAIN THAT
 273 REM   THE ARRAY BOUNDS ARE NEVER OUT OF BOUNDS WITH A VALUE
 274 REM   OF 26  (30 CHOSEN AS A CONVIENENT NUMBER).
 275 REM
 280 DIM A[30],B[30]
 290 REM
 291 REM
 292 REM  ARRAY VALUES FOR A
 293 REM
 300 DATA 6.50000E-[illegible],6.80000E-02,1,1.25,1.4
 320 DATA 1.5,1.5,1.5,[illegible]
 340 DATA 6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03
 360 DATA 6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03
 380 DATA 6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03,6.60000E-03
 400 DATA 6.50000E-[illegible],[illegible]E-02,1,1.25,1.4
 410 REM
 411 REM
 412 REM  ARRAY VALUES FOR B
 413 REM
 420 DATA 3.30000E-03,5.00000E-02,.1,.15,.17
 440 DATA .24,.3,.36,.4,.3
 460 DATA 3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03
 480 DATA 3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03
 500 DATA 3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03,3.30000E-03
 520 DATA 3.30000E-03,5.00000E-02,.1,.15,.17
 [illegible] REM
 [illegible] REM
 [illegible] REM  READ THE ARRAYS A AND B WITH A MAT READ STATEMENT
 [illegible] REM
 [illegible] MAT  READ A
 [illegible] MAT  READ B
 [illegible] REM
 [illegible] REM
 [illegible] REM
 [illegible] REM  [illegible]
 [illegible] REM

      PAGE  0002          *PHYS3          19 MAY 1978   00:10:12

 580 DATA 100,120,150,150,3340,500,240
 600 READ V1,V2,V3,V4,V5,V6,V7
 620 DATA 0,0,0,0,0,0,0
 640 READ F1,F2,F3,F4,F5,F6,F7
 660 DATA 0,0,0,0,0,0,0
 680 READ P1,P2,P3,P4,P5,P6,P7
 690 REM
 691 REM
 692 REM  SET THE DISPLAY HEIGHT AND DEPTH TO 150
 693 REM
 700 LET @[3]=150
 720 LET @[4]=-@[3]
 730 REM
 731 REM
 732 REM  SET THE DISPLAY LENGTH FOR 5
 733 REM
 740 LET @1=5
 750 REM
 751 REM
 752 REM  SET THE MAXIMUM RELATIVE ERROR TOLERANCE TO 0.002
 753 REM  SET THE MINIMUM STEP SIZE TO 0.001
 754 REM
 760 LET @5 = 0.002
 770 LET @7 = 0.001
 [illegible] DYNAMIC
 800  EQUATIONS
 [illegible]   INDVAR T
 [illegible]   REM
 [illegible]   REM
 832   REM  THE NEXT THREE PROCEDURE BLOCKS CALCULATE THE HEART PUMP
 833   REM  LOOKUP VALUES AND INTERPOLATE THE TABLE.  THE FIRST BLOCK
 834   REM  DETERMINES THE ARRAY INDEX  K.  THE SECOND AND THIRD PERFORM
 835   REM  THE INTERPOLATION YEILDING A1 FROM A  AND B1  FROM B.
 836   REM
 840   PROCED K=T
 860    LET K=(T-INT(T))*25+1
 865   PROEND
 870   PROCED A1=K
 880    LET C=K+.5-INT(K+.5)
 900    LET A1=A[K]+(A[K+1]-A[K])*C
 905   PROEND
 910   PROCED B1=K
 915    LET C=K+.5-INT(K+.5)
 920    LET B1=B[K]+(B[K+1]-B[K])*C
 940   PROEND
 950   REM
 951   REM
 952   REM  THE FOLLOWING PROCEDURE BLOCKS ALL PERFORM THE MAX FUNCTION
 953   REM  WHICH THIS BASIC DOES NOT HAVE.  UNDER EACH HEADING THE
 954   REM  EQUATIONS AND PROCEDURE BLOCKS ARE ENTERED.
 955   REM
 960   REM    PULMONARY VENIAL SYSTEM
 980    P7=3.30000E-02*V7
1000    F4=7*(P2-P7)

      PAGE  0003          *PHYS3          19 MAY 1978   00:10:12

1010  DER V7=F4-F3
1020   REM  LEFT VENTRICAL
1040    P3=V3*A1
1060   PROCED F3=P7,P3
1080    LET H=P7-P3
1100    IF H>0 THEN 1160
1120    LET F3=0
1140    GOTO 1180
1160    LET F3=17*H
1180   PROEND
1190  DER V3=F3-F2
1200   REM  AORTA
1220    P1=.8*V1
1240   PROCED F2=P3,P1
1260    LET H=P3-P1
1280    IF H>0 THEN 1340
1300    LET F2=0
1320    GOTO 1360
1340    LET F2=80*H
1360   PROEND
1370   DER V1=F2-F1
1380   REM    SYSTEMIC CIRCULATION
1400    P5=1.53000E-02*V5
1420    F1=1.63*(P1-P5
1430   DER V5=F1-F7
1440   REM  VENA CAVA
1460    P6=4.00000E-03*V6
1480    F7=1.65*(P5-P6)
1490   DER V6=F7-F6
1500   REM   RIGHT VENTRICAL
1520    P4=V4*B1
1540   PROCED F6=P6,P4
1560    LET H=P6-P4
1580    IF H>0 THEN 1640
1600    LET F6=0
1620    GOTO 1660
1640    LET F6=78*H
1660   PROEND
1665   DER V4=F6-F5
1670   REM  PULMONARY ARTERIAL SYSTEM
1680    P2=.133*V2
1700   PROCED F5=P4,P2
1720    LET H=P4-P2
1740    IF H>0 THEN 1800
1760    LET F5=0
1780    GOTO 1820
1800    LET F5=90*H
1820   PROEND
1840   DER V2=F5-F4
1900   REM
1910   REM
1920   REM   DISPLAY PAO AND PLV
1930   REM
2000   DISPLAY P1,P3

      PAGE  0004          *PHYS3          19 MAY 1978   00:10:12

2020  EQUEND
2040 DYEND
2060 END
```


Downloaded from https://academic.oup.com/comjnl/article/23/3/233/375185 by guest on 10 April 2024

Table 2 Comparison of execution times (based on Korn and Wait, 1976)

System	*Computer*	*Integration Routine*	*Execution Time (sec)*
(*a*) Batch-Processed, Equation-Oriented, Floating-Point Languages, Large Computers			
FORTRAN	CDC 3600	Not known	2
CSMP III	IBM 360/50	Not known	1·46
	IBM 360/65	Not known	0·42
DARE P	CDC 6400	Runge-Kutta 2	0·16
		Runge-Kutta-Merson (0·001 rel. error)	0·91
RSSL	CDC 6600	Runge-Kutta 2	0·056
(*b*) Interactive, Equation-Oriented, Floating-Point Language, Minicomputers			
DARE/ELEVEN	DEC PDP-11/40		
	Floating-point software (DOS)	Runge-Kutta 2	6·5
	Floating-point firmware (RT-11)	Runge-Kutta 2	3·6
	DEC PDP-11/45		
	Core memory, floating-point software (DOS)	Runge-Kutta 2	4·5
	Bipolar memory, floating-point hardware	Runge-Kutta 2	< 1·2* estimated
(*c*) Interactive, Block-Diagram, Floating-Point Languages, Minicomputers			
Block CSMP	IBM 1130		
	Interpreter mode	Runge-Kutta 2	40
	Compiler mode	Runge-Kutta 2	20
	DEC PDP-11/45	Not known	14·6
	FP software only		
ISL-11†	DEC PDP-11/20	Euler	1·4
	DEC PDP-11/45	Runge-Kutta 2	2·8
	FP software only	Euler	0·6
(*d*) Interactive, Block-Diagram, Fixed-Point Languages, Minicomputers (with hardware multiplication)			
DARE II	DEC PDP-9	Runge-Kutta 2	0·9
DARE/ELEVEN	DEC PDP-11/40	Runge-Kutta 2	0·19
	DEC PDP-11/45 bipolar memory	Runge-Kutta 2	0·06 estimated
MICRODARE	DEC PDP-11/40	Runge-Kutta 2	0·3
	DEC PDP-11/03	Runge-Kutta 2	0·9
(*e*) Interactive, Equation-Oriented, Fixed-Point Language with Semiautomatic Scaling, Minicomputers (with hardware multiplication)			
SIMEX	DEC PDP-9	Euler	0·19
	DEC PDP-15	Euler	0·15 estimated
BDARE (BASIC)	PDP-11/40 no FP firmware	Runge-Kutta 2	31
BEDSOCS	HP 2100 with FP	Runge-Kutta Merson 4th order	22

region is split into two sections, a procedural part and the representation set.

When control flows into the dynamic region, it encounters the first procedural section. This region (transparently to the user) repeats derivative calls and integration steps as determined by the derivative computing routine, defined by procedure blocks and equations within the representation set. This routine is executed until a communication point is encountered. Such points occur at regular intervals set by the user in the control region, or at points determined by test values described by 'EXIT' statements. When a communication point occurs, execution is transferred to a procedural section of the dynamic region following the equations section of the program, typically to produce output. **Fig.** 3 shows a program which uses the communication points by means of the 'EXIT' statement. When the value of the inequality in the 'EXIT' statement changes from 'FALSE' to 'TRUE', the program transfers control to the procedural section following the representation set as shown.

BEDSOCS uses a 4th order Runge-Kutta-Merson integration routine which allows variable step sizes. By using control variables, the user can set the error tolerance and the minimum step size. As the simulation continues, the step size is adjusted according to the estimated truncation error.

In a variable step integration, the user does not choose the values of the independent variable at which solutions are evaluated. For example, Fig. 3 shows a program which finds the initial angle and velocity necessary for a projectile to pass over a wall and hit a target on the other side. For this procedure it is necessary to know the altitude of the projectile as it passes over the wall. The 'EXIT' statement in line 250 causes the integration routine to back up and hit the point where *X* becomes

Downloaded from https://academic.oup.com/comjnl/article/23/3/233/375185 by guest on 10 April 2024

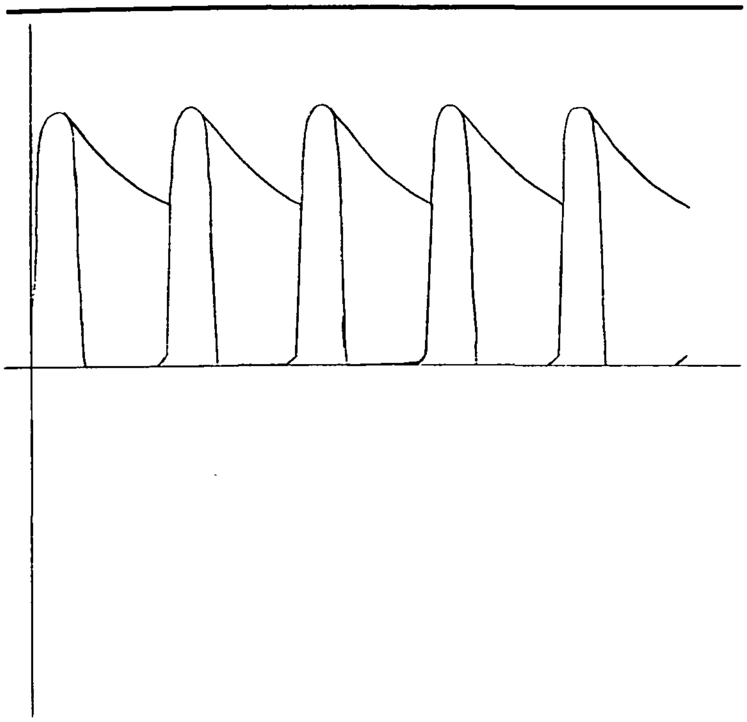

Fig. 5

greater than or equal to W. In this way particular points may be evaluated while areas of lesser interest may be passed over using larger integration steps.

The BEDSOCS display system is completely automatic. To display the variable X as it is evaluated, it is only necessary to place a '*DISPLAY X*' statement in the equation section of the program. This will display X as a function of the independent variable as the simulation is performed. The axis values are generally set in the control region of the program, but they may be dynamically altered through the communication (procedural) sections or the procedure blocks. BEDSOCS automatically draws the axes and plots a multiplicity of lines. For example, '*DISPLAY T*; *A*, *B*, *C*' draws the variables A, B, and C against the variable T throughout the simulation. Although only one display statement may be active at a time, special '*USE*' statements allow a choice among several display statements. Each statement is activated at a different time during the execution.

In order to measure the execution times for BEDSOCS programs, PHYSBE, a well known benchmark program (Korn and Wait, 1976), was used. This program simulates the human circulatory system. **Fig. 4** contains a diagram of the seven differential equations describing blood flow. The system is driven by the two chambers of the heart, represented by a table of values (**Table 1**). **Table 2** shows how BEDSOCS compares with other simulation systems in time required to execute PHYSBE. Although BEDSOCS is comparatively slow (execution time of 22 seconds per heart beat), its interpretive features greatly reduce development time. **Fig. 5** is a sample PHYSBE execution.

Perhaps the most significant feature of BEDSOCS is the ease with which it can be mastered. Using BASIC as a procedural language and implementing the representation set in a familiar format allows even the novice to write complex simulation programs. BEDSOCS not only requires no compilation time, but also greatly simplifies debugging. For users requiring immediate response on small problems, BEDSOCS is an attractive simulation language.

Acknowledgement

We wish to acknowledge the assistance given by the University of Arizona Electrical Engineering Department, particularly on course EE370.

References

Brown, G. (1973). BEDSOCS MARK I Reference Manual, Postgraduate School of Studies in computing, University of Bradford, August 1973.

Korn, G. A. and Wait, J. V. (1976.) *Digital Continuous-Systems Simulation*, Prentice-Hall, Englewood Cliffs, NJ.

Book review continued from page 229

oriented processing system (CHEOPS) illustrates the seminal influence of AI on computer science. It describes special hardware to deal with the search part of chess programs, it makes practical a depth of 8 or 9 while experimenting with different heuristic strategies. The special hardware is front ended by conventional computers to relieve it of I/O and housekeeping. The short paper includes a description of the facilities used to design and fabricate the hardware. There are 16 accumulators, a 1024 words pushdown list, a chess array module to carry out as single instructions standard checks, moves and other operations; microcode is used. The paper by Arlazarov and Futer is concerned with the king rook pawn against king rook end game. Data structures are developed to minimise storage and processing, symmetries and other properties are used to cut down further the number of cases (about 10^9). Starting with the end positions the resulting reduced set is generated iteratively and classified into win or draw with the appropriate move. The paper by Adelson-Velsky, Arlazarov and Donskoy on algorithms of adaptive search in computer chess uses influence relations to construct mathematical expressions in terms of graphs and trees for various situations such as attack, pin, check, etc. The results of various searches are stored using appropriate data structures.

The first paper in the section on knowledge engineering is by Buchanan on issues of representation in conveying the scope and limitations of intelligent assistant programs. The paper starts with a brief description of DENDRAL, the famous expert system to aid chemists in determining chemical structures. The main modules include CONGEN which generates a large number of possible structures consistent with the given chemical and spectroscopic data, the planner which interacts with the chemist to make use of his expert knowledge, and the meta-dendral which makes use of the accumulated structure-spectrum data to formulate new rules to explain the behaviour of compounds. The paper concentrates on the internal representation most suitable in communicating with the users and experts, documentation issues, and communicating the progress of the computation in the form of a laboratory notebook including a trace of the reasoning steps. The second paper by Briabrin describes the internal structure of DILOS, the system described in the paper by Pospelov and Pospelov above. It consists of a set of LISP programs to set up and search a data base, initiation of application programs, logical analysis of the problem area, and consistency checking of new facts added to the data base.

The final section is on natural language and the first paper on natural language for interaction with a data base by Senin is also on DILOS. It is concerned with the linguistic processor module which developed the ϕ-language for user communication with the system. The paper by Narin'jani describes the AI work in the Siberian branch of the USSR academy of sciences. This concentrates on the nondeterministic behaviour model in robot control and formal models, semantic representation and interrogation methods in the area of natural languages. The final paper is on purposive understanding by Schank and DeJong. It starts with a short and clear review of the various stages of developing the SAM computer system to understand natural language, then describes in more detail a new system, FRUMP, which concentrates in a top down fashion on selective rather than total understanding.

I. M. Khabaza (London)

Downloaded from https://academic.oup.com/comjnl/article/23/3/233/375185 by guest on 10 April 2024